

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appl. No.	: 10/677,050	Group Art Unit 2154
Applicants	: Mohamed K. Diab et al.	
Filed	: September 30, 2003	
TC/A.U.	: 3768	
Examiner	: Eric F. Winakur	
Title	: SIGNAL PROCESSING APPARATUS	
Docket No.	: MASIMO.7CP1C10	
Customer No.	: 20,995	

DECLARATION UNDER 37 C.F.R. § 1.131

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

1. This declaration is to establish the status of the invention in the above-captioned U.S. Patent Application prior to March 26, 1993, the filing date of U.S. Patent No. 5,368,026, issued on November 29, 1994, to Swedlow et al. ("Swedlow").

2. We understand that Claims 1-10, 12-16, 18-25, 27-36, and 38-46 of the above-captioned U.S. Patent Application Ser. No. 10/677,050 (the '050 Patent Application), which was filed September 30, 2003, titled "*Signal Processing Apparatus*," have been rejected in the February 27, 2008 Office Action as being unpatentable over Swedlow.

3. We are inventors of the subject matter of the above-referenced claims in the '050 Patent Application.

4. At least prior to March 26, 1993, we conceived of and reduced to practice a signal processing apparatus such as the device and method described and claimed in the above-referenced claims of the '050 patent application.

5. As evidence thereof, we provide herewith **Exhibit A** which is a design note ("the Design Note") describing the subject matter of the above-referenced claims. The Design Note was created and is dated prior to March 26, 1993. We conceived the

ideas referenced therein prior to the creation of the Design Note. Any references to dates on the Design Note have been redacted.

6. At Masimo Corporation ("Masimo"), the term "Stork" refers generally to Masimo's pulse oximetry technologies.

7. At least prior to March 26, 1993, we compiled, loaded onto hardware, and used a software application called "STORK" for its intended purpose, that is, to measure blood oxygen saturation.

8. As evidence thereof, we provide herewith **Exhibits B** and **C** which are screen captures of a directory of files, including ".c" source code files, ".h" header code files, and ".gdd" debugging code files compiled into the application file STORK. As shown on the screen captures, each of these files includes a last edited date prior to March 26, 1993, which has been redacted herein.

9. **Exhibits D** and **E** are the header and source code files, titled "*vector.h*," and "*vector.c*," respectively, written in the C programming language, showing the mathematical functions for calculating a vector cross correlation.

10. **Exhibits F** and **G** are the header and debugging files, titled "*stork.h*," and "*stork.gdd*," respectively, written in the C programming language. The header file defines the variable for cross correlation and the debugging file displays the value of cross correlation for instructive purposes.

11. **Exhibit H** is the source code file, titled "*iawa.c*," written in the C programming language, which calculated the cross correlations.

12. **Exhibit I** is the source code file, titled "*satfil.c*," written in the C programming language, which qualifies bins of signal data responsive to intensity signals from a detector monitoring a patient, the qualification including determining whether the calculated cross correlation is within expected limits.

13. **Exhibit J** is the source code file, titled "*ui.c*," written in the C programming language, which is the user interface for the STORK application and which indicates an expected limit of cross correlation.

14. The foregoing files and other exhibits demonstrate that we constructed and tested an embodiment and performed processes meeting every element of the

above-referenced claims prior to March 26, 1993 sufficient to demonstrate that the constructed embodiment operated for its intended purpose.

15. We hereby declare that acts leading to the reduction to practice of the subject matter claimed in the '050 Patent Application were performed in the United States.

I declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and further that these statements were made with the knowledge that willful, false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful, false statements may jeopardize the validity of the application or any patent resulting therefrom.

Dated: _____

By: _____

Mohamed K. Diab

I declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and further that these statements were made with the knowledge that willful, false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful, false statements may jeopardize the validity of the application or any patent resulting therefrom.

Dated: _____

By: _____

Walter M. Weber

I declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and further that these statements were made with the knowledge that willful, false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful, false statements may jeopardize the validity of the application or any patent resulting therefrom.

Dated: August 18, 2008

By: 
Massi E. Kiani

Exhibit A

EXHIBIT A

MASIMO™		CORRELATION DETECTION ALGORITHM			Design Notes
Design Note 37	Revision A	File SDN37A.DOC	Project Stork	Page 1 of 2	
Author Walter Weber	Revision Date	Original Date			

The goal of this note is to propose an artifact detector for the Stork algorithm. Specifically, the artifact detector is designed such that it assigns a real number to each snapshot of data that it processes, and then compares that real number to a predetermined threshold value. Snapshots of data which yield artifact detector values that lie below the threshold value are said to contain significant amounts of artifact. Snapshots of data that yield artifact detector values which lie above the threshold value are said to contain insignificant amounts of artifact.

In principle, the Stork algorithm accepts as input two signals which are of the form (after log and bandpass)

$$\begin{aligned} S_a(t) &= s_a(t) + n_a(t) \\ S_b(t) &= s_b(t) + n_b(t) \end{aligned} \quad (1)$$

where

$$\begin{aligned} s_b(t) &= \omega_1 s_a(t) \\ n_b(t) &= \omega_2 n_a(t). \end{aligned} \quad (2)$$

Any input signals which are not expressible in the above form are said to consist of artifacts. Consequently to determine whether or not a signal is artifact free, it is sufficient to determine whether or not equations (1) and (2) are satisfied.

One way to check whether or not equations (1) and (2) have been met is to measure the correlation between $S_a(t)$ and $S_b(t)$. Specifically one could compute the cross correlation

$$\int S_a(t) S_b(t) dt = \int (s_a(t) s_b(t) + n_a(t) n_b(t)) dt + \int (s_a(t) n_b(t) + s_b(t) n_a(t)) dt$$

which assuming that the signals, $s_a(t)$ and $s_b(t)$, are uncorrelated with the noises, $n_a(t)$ and $n_b(t)$, may be expressed as

$$\int S_a(t) S_b(t) dt = \int (s_a(t) s_b(t) + n_a(t) n_b(t)) dt.$$

A normalized cross correlation may be computed as follows

$$\frac{\int S_a(t) S_b(t) dt}{[\int S_a^2(t) dt \int S_b^2(t) dt]^{0.5}} = \frac{\int (s_a(t) s_b(t) + n_a(t) n_b(t)) dt}{[\int (s_a^2(t) + n_a^2(t)) dt \int (s_b^2(t) + n_b^2(t)) dt]^{0.5}}$$

The solutions which give the extrema of this cross correlation functional are given by equation (2). Any input signals which are not expressible in the form given in equation (2) yield reduced normalized correlation values. In Figure 1, these extrema of the normalized correlation functional are plotted vs. the

"Red" Signal - to - Noise ratio and ω_1/ω_2 . One should note that the normalized correlation is not necessarily unity and precaution must be taken, particularly when the difference between arterial saturation and the background saturation is large.

One may rigorously prove that the magnitude of the normalized cross correlation is always less than or equal to unity by use of the inner product

$$x \cdot y = \int x(t) y(t) dt.$$

Using the above notation, the normalized cross correlation may be written as

$$\frac{s_a \cdot s_b + n_a \cdot n_b}{[(s_a \cdot s_a + n_a \cdot n_a)(s_b \cdot s_b + n_b \cdot n_b)]^{0.5}}$$

or equivalently

$$\frac{S_a \cdot S_b}{[S_a \cdot S_a S_b \cdot S_b]^{0.5}}.$$

But, according to the Schwarz Inequality

$$(S_a \cdot S_b)^2 \leq (S_a \cdot S_a)(S_b \cdot S_b).$$

Hence the normalized correlation is always between 1 and -1.

ALGORITHM IMPLEMENTATION

1. Compute the quantity

$$\frac{\sum S_a S_b}{[\sum S_a^2 \sum S_b^2]^{0.5}}$$

2. Compare quantity to a threshold, say 0.75

THINGS THAT STILL NEED TO BE INVESTIGATED

Investigate performance vs. snapshot size

Investigate performance vs signal/noise correlation

NORMALIZED CROSS CORRELATION
VERSUS η/η_0 AND RED SKEW

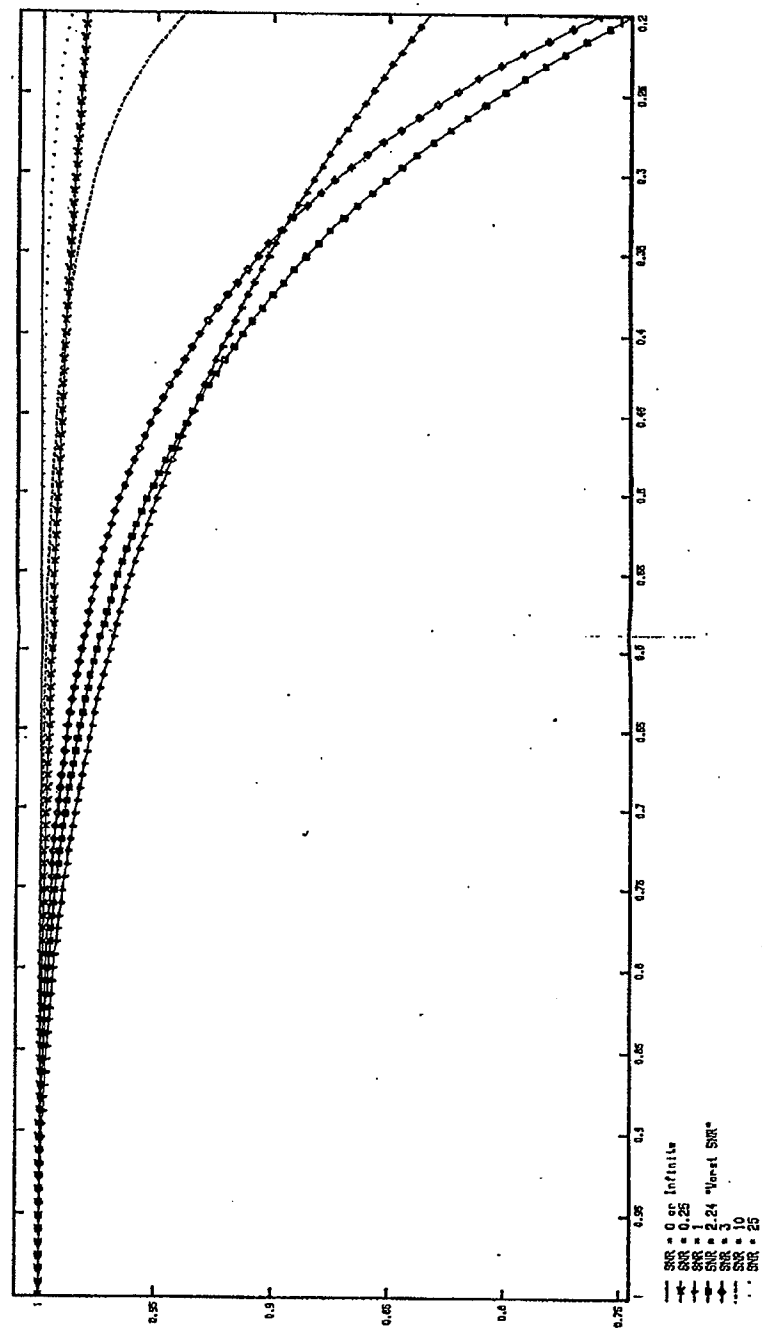


Exhibit B

C:\Masimo\Patent Issues\Correlation\R154B					
File Edit View Favorites Tools Help					
Back Forward Stop Search Folders					
Address C:\Masimo\Patent Issues\Correlation\R154B Go					
File and Folder Tasks	Name	Size	Type	Date Modified	
Other Places	HOWAYGEN.H	2 KB	C Header file	2/19/1993 11:27 AM	
Details	HOWAYGEN.C	3 KB	C Source file	2/19/1993 11:40 AM	
	Z1WAYGEN.C	4 KB	C Source file	2/19/1993 11:45 AM	
	LSL.H	3 KB	C Header file	2/23/1993 12:53 PM	
	STORK.C	4 KB	C Source file	2/23/1993 1:26 PM	
	GDDCL.H	4 KB	C Header file	2/23/1993 3:15 PM	
	DISPLAY.C	12 KB	C Source file	2/23/1993 5:41 PM	
	GDDCL.C	6 KB	C Source file	2/23/1993 5:59 PM	
	ADC.H	2 KB	C Header file	2/24/1993 3:02 PM	
	ADC.C.C	3 KB	C Source file	2/24/1993 3:03 PM	
	HW.H	3 KB	C Header file	2/25/1993 12:00 PM	
	CALIB.C	10 KB	C Source file	2/25/1993 1:00 PM	
	VECTOR.H	3 KB	C Header file	2/25/1993 4:26 PM	
	VECTOR.C	5 KB	C Source file	2/25/1993 4:26 PM	
	FFT.H	4 KB	C Header file	2/26/1993 5:50 PM	
	FFT.C	12 KB	C Source file	3/2/1993 10:09 AM	
	OPTANC.C.C	7 KB	C Source file	3/2/1993 1:12 PM	
	VSIHR.H	2 KB	C Header file	3/2/1993 2:45 PM	
	VSIPL.C	4 KB	C Source file	3/2/1993 2:47 PM	
	VSIPL.H	2 KB	C Header file	3/2/1993 2:47 PM	
	FR.C.C	4 KB	C Source file	3/2/1993 5:51 PM	
	LSL.C.C	4 KB	C Source file	3/3/1993 5:03 PM	
	SPT.H	2 KB	C Header file	3/3/1993 5:03 PM	
	VSIHR.C	13 KB	C Source file	3/3/1993 5:28 PM	
	HRFIL.C	5 KB	C Source file	3/5/1993 2:21 PM	
	ULH	1 KB	C Header file	3/5/1993 2:54 PM	
	CALIB.H	4 KB	C Header file	3/5/1993 3:06 PM	
	VSIIDUAL.H	2 KB	C Header file	3/5/1993 3:06 PM	
	SATEQN.C	4 KB	C Source file	3/5/1993 3:27 PM	
	VSIIDUAL.C	4 KB	C Source file	3/5/1993 3:37 PM	
	MAKEFILE	7 KB	File	3/6/1993 1:36 PM	
	Z1PEAK.C	3 KB	C Source file	3/6/1993 1:41 PM	
	PEAK.C	6 KB	C Source file	3/8/1993 8:58 AM	
	USR1.H	3 KB	C Header file	3/8/1993 10:21 AM	
	STORK.GDD	9 KB	GDD File	3/8/1993 12:47 PM	
	STORK.H	10 KB	C Header file	3/8/1993 12:50 PM	
	VSIINC.C	14 KB	C Source file	3/8/1993 1:04 PM	
	SATFIL.C	10 KB	C Source file	3/8/1993 1:07 PM	
	GDD1	0 KB	DAT File	3/8/1993 1:15 PM	
	DEBUG	0 KB	Text Document	3/8/1993 2:06 PM	
	JAWA.C	6 KB	C Source file	3/8/1993 3:44 PM	
	ULC	30 KB	C Source file	3/8/1993 3:45 PM	
	USR1.C.C	14 KB	C Source file	3/8/1993 3:48 PM	
	STORK	260 KB	Application	3/8/1993 3:49 PM	

Exhibit C

C:\Masimo\Patent Issues\Correlation\RTS4B				
File Edit View Favorites Tools Help				
Back Forward Stop Search Folders				
Address C:\Masimo\Patent Issues\Correlation\RTS4B Go				
File and Folder Tasks	Name	Size	Type	Date Modified
Other Places	ASH_SPRT.H	3 KB	C Header file	1/12/1992 2:43 PM
Details	DEF21020.H	8 KB	C Header file	7/21/1992 11:27 AM
	NANC.H	1 KB	C Header file	11/6/1992 5:33 PM
	STRKLOG.H	1 KB	C Header file	11/10/1992 11:29 AM
	VSINC.H	2 KB	C Header file	12/8/1992 3:55 PM
	STATS.H	1 KB	C Header file	12/30/1992 4:53 PM
	PEACH	2 KB	C Header file	12/30/1992 4:54 PM
	SATFIL.H	1 KB	C Header file	1/16/1993 2:34 PM
	HRFIL.H	2 KB	C Header file	1/18/1993 2:38 PM
	PAVEQ.H	2 KB	C Header file	1/18/1993 5:11 PM
	SATEQN.H	2 KB	C Header file	1/25/1993 2:14 PM
	PACKET.H	Picture (Device Independent Bitmap) header file		2/3/1993 4:29 PM
	POST.H	1 KB	C Header file	2/4/1993 11:22 AM
	NANC.C	7 KB	C Source file	2/9/1993 2:23 PM
	STATS.C	3 KB	C Source file	2/9/1993 2:24 PM
	PACKET.C	5 KB	C Source file	2/9/1993 2:28 PM
	NELLCOR.H	2 KB	C Header file	2/12/1993 4:40 PM
	NELLCOR.C	3 KB	C Source file	2/12/1993 4:52 PM
	FIL.H	3 KB	C Header file	2/16/1993 2:37 PM
	ZFIR.C.C	3 KB	C Source file	2/16/1993 2:38 PM
	POSTC6.C.C	8 KB	C Source file	2/16/1993 2:45 PM
	ZPOSTC6.C	8 KB	C Source file	2/16/1993 2:45 PM
	OPTANCH	1 KB	C Header file	2/16/1993 2:50 PM
	CD.F.H	2 KB	C Header file	2/17/1993 10:32 AM
	ZUSRL.C.C	6 KB	C Source file	2/17/1993 11:27 AM
	CD.F.C	4 KB	C Source file	2/17/1993 2:30 PM
	ZZZULC	23 KB	C Source file	2/18/1993 3:11 PM
	ZGDOCLIC	6 KB	C Source file	2/18/1993 5:45 PM
	ZICALJB.C	1 KB	C Source file	2/18/1993 6:53 PM
	Z1VSINC.C	2 KB	C Source file	2/18/1993 6:56 PM
	Z1NANC.C	1 KB	C Source file	2/18/1993 6:57 PM
	Z1SATFIL.C	1 KB	C Source file	2/18/1993 6:58 PM
	Z1STATS.C	2 KB	C Source file	2/18/1993 6:59 PM
	Z1OPTANC.C	3 KB	C Source file	2/18/1993 7:02 PM
	Z1VSIHR.C	1 KB	C Source file	2/18/1993 7:04 PM
	Z1VSIPL.C	1 KB	C Source file	2/18/1993 7:06 PM
	Z1HRFIL.C	1 KB	C Source file	2/18/1993 7:08 PM
	Z1CDF.C	1 KB	C Source file	2/18/1993 7:10 PM
	Z1LSLC	1 KB	C Source file	2/18/1993 7:13 PM
	Z1SATEQN.C	2 KB	C Source file	2/18/1993 7:17 PM
	Z1IAWA.C	1 KB	C Source file	2/18/1993 7:19 PM
	Z1ADC.C	1 KB	C Source file	2/18/1993 7:20 PM
	Z1PACKET.C	1 KB	C Source file	2/18/1993 7:22 PM
	Z1NELCOR.C	1 KB	C Source file	2/18/1993 7:26 PM

Exhibit D

```

1  /* -----
   Copyright (c) Masimo Corporation (tm) 1993. All Rights Reserved.

   File: vector.h

5  Description:
      Collection of routines that work on vectors. Designed for
      Masimo Standard Library.

10  Public Functions:
      min_index      Returns index to minimum element in vector
      max_index      Returns index to maximum element in vector
      inner_product  Returns point by point product of vectors
      remove_mean    Removes the mean value from each point in a vector

15  VECTOR_DotProduct    Computes Dot product of two vectors
      VECTOR_CrossCorr0  Computes Correlation Coefficient

      Private Functions:

20  Notes:

      History:

25  01/15/93      RJM      Initial Version
      01/25/93      RJM      add remove_mean
      02/25/93      RJM      add VECTOR_DotProduct
      02/25/93      RJM      Add VECTOR_CrossCore0

30  ----- */

#include <masimo.h>

extern INT32
35  min_index(
      FLOAT32      *data,      /* input, pointer to data      */
      INT32        size);      /* input, number of data points */

extern INT32
40  max_index(
      FLOAT32      *data,      /* input, pointer to data      */
      INT32        size);      /* input, number of data points */

extern VOID
45  inner_product(
      FLOAT32      *v1,      /* input, first vector      */
      FLOAT32      *v2,      /* input, second vector     */
      INT32        size,      /* input, size of vectors   */
      FLOAT32      *v3);      /* output, v3 = v1 * v2     */

50  extern VOID
      remove_mean(
      FLOAT32      *in,      /* input, data      */
      FLOAT32      *out,      /* output, data with mean removed */
      INT32        size);      /* input, number of points */

55  extern FLOAT32
      VECTOR_DotProduct(
      FLOAT32      *u,      /* input, first vector      */
      FLOAT32      *v,      /* input, second vector     */
      INT32        size);      /* input, size of vectors   */

60  extern FLOAT32
      VECTOR_CrossCorr0(
      FLOAT32      *u,      /* input, first vector      */
      FLOAT32      *v,      /* input, second vector     */
      INT32        size);      /* input, size of vectors u and v */

65

```

Exhibit E

```

1  /* -----
   Copyright (c) Masimo Corporation (tm) 1993. All Rights Reserved.

   File: vector.c

5  Description:
   Collection of routines that work on vectors. Designed for
   Masimo Standard Library.

10  Public Functions:
   min_index      Returns index to smallest element in vector
   max_index      Returns index to largest element in vector
   inner_product  Computes the inner product of two vectors
   remove_mean    Removes the mean value from each point in a vector
15  VECTOR_DotProduct  Compute Dot Product of two vectors
   VECTOR_CrossCorr0  computes cross correlation time=0

   Private Functions:

20  Notes:

   History:

       01/15/93  RJM  Initial Version
       02/25/93  RJM  Add VECTOR_DotProduct
       02/25/93  RJM  Add VECTOR_CrossCorr0
   ----- */

30  #define MODULE_ID      110

   #include <masimo.h>          /* platform definitions      */
   #include <math.h>            /* sqrt function            */
   #include "vector.h"         /* self                     */
35

   INT32
   min_index(
       FLOAT32      *data,      /* input, pointer to data    */
       INT32        size) {     /* input, number of data points */

       INT32        i;
       INT32        j;
       FLOAT32      tmp;
45

       BUG1(data); BUG1(size);

       j = 0;
       tmp = data[j];
50       for (i = 1; i < size; i++) {
           if (data[i] < tmp) {
               tmp = data[i];
               j = i;
           }
55       }
       return(j);
   }

60  INT32
   max_index(
       FLOAT32      *data,      /* input, pointer to data    */
       INT32        size) {     /* input, number of data points */

       INT32        i;
       INT32        j;
       FLOAT32      tmp;
65

```

```

84     BUG1(data); BUG1(size);
70
    j = 0;
    tmp = data[j];
    for (i = 1; i < size; i++) {
        if (data[i] > tmp) {
75             tmp = data[i];
                j = i;
        }
    }
    return(j);
80
}

VOID
85 inner_product(
    FLOAT32    *v1,          /* input, first vector      */
    FLOAT32    *v2,          /* input, second vector     */
    INT32      size,         /* input, size of vectors   */
    FLOAT32    *v3) {        /* output, v3 = v1 * v2    */
90
    INT32      i;

    for (i = 0; i < size; i++) {
95        v3[i] = v1[i] * v2[i];
    }
}

VOID
100 remove_mean(
    FLOAT32    *in,          /* input, data              */
    FLOAT32    *out,         /* output, data with mean removed */
    INT32      size) {       /* input, number of points   */
105
    INT32      i;
    FLOAT32    mean;

    BUG1(in); BUG1(out); BUG1(size);

110    mean = 0.0;
    for (i = 0; i < size; i++) mean += in[i]; /* sum all points */
    mean /= (FLOAT32)size;
    for (i = 0; i < size; i++) out[i] = in[i] - mean;
115 }

FLOAT32
VECTOR_DotProduct(
120    FLOAT32    *u,          /* input, first vector      */
    FLOAT32    *v,          /* input, second vector     */
    INT32      size) {       /* input, size of vectors   */
125
    FLOAT32    sum;
    INT32      i;

    BUG1(u); BUG1(v); BUG1(size);

    sum = 0.0;
130    for (i = 0; i < size; i++) {
        sum += u[i] * v[i];
    }
    return(sum);
135 }

```

```

    FLOAT32
    VECTOR_CrossCorr0(
140      FLOAT32      *u,          /* input, first vector      */
        FLOAT32      *v,          /* input, second vector     */
        INT32        size) {      /* input, size of vectors u and v */

        FLOAT32      den;
145      FLOAT32      c;

        BUG1(u); BUG1(v); BUG1(size);

        den = SQRTF(VECTOR_DotProduct(u, u, size) *
150          VECTOR_DotProduct(v, v, size));

        if (den == 0.0) {
            c = 1.0;
        } else {
155          c = VECTOR_DotProduct(u, v, size) / den;
        }
        return(c);
    }

```

Exhibit F

```

1  /* -----
   Copyright (c) Masimo Corporation (tm) 1992, 1993. All Rights Reserved.

   File: stork.h

5  Description: Stork header file included by ALL stork modules

   Public Functions:

10  Private Functions:

   Notes:

   History:

15      RJM      11/06/92      Initial version (v1.29)
      RJM      12/08/92      Prevent multiple inclusion of stork.h
      RJM      12/16/92      Change vsinc params to all floats
      RJM      01/04/93      Add MAX_BINS
20      RJM      01/05/93      Add seed to bin structure
      RJM      01/14/93      Put conditional code around "C" only stuff
      RJM      01/15/93      Make memory pool sizes platform dependant
      RJM      01/16/93      Change sat result structure for filtering
      RJM      01/18/93      Make HR a vector
25      RJM      01/18/93      Add more sat params
      RJM      01/19/93      Add hr_max_delta and sat_max_delta params
      RJM      01/25/93      Make SAT equation adjustable
      RJM      01/25/93      Make HR algorithms switchable

30  ----- */

#ifndef STORK
#define STORK

35  #define VERSION      1.541  /* floating point software version number */

#define VSINC_LEAN      0      /* Generate code for Lean Architecture */

40  #define ADC_SNAPSHOT_SIZE      64

/* ----- */
/*                      Memory Allocation                      */
/* ----- */

45  #if (PLATFORM == 1)

#define MEM_STACK_SIZE      48
#define MEM_POOL_SIZE      32768
#define DEBUG_MEM_SIZE      512
50  #define GDD_BUF_SIZE      20000
#define DBG_FIFO_SIZE      100

#else

55  #define MEM_STACK_SIZE      48      /* data memory */
#define MEM_POOL_SIZE      26000      /* data memory */
#define DEBUG_MEM_SIZE      512      /* data memory */
#define GDD_BUF_SIZE      6000      /* pmda (was 6000) */
60  #define DBG_FIFO_SIZE      100      /* pmda */

#endif

/* The following constant controls ALL debug code for the STORK */
/* beta software. No required code should be conditioned on this */
65  /* constant. */

#define DEBUG 1      /* enables debug code */

```

```

70  /* -----
    The following constants control what data is captured in the dbg_fifo.
    The dbg_fifo is controlled by the ADC interrupt routine.
    ----- */

75  #define DEBUG_SIM      0          /* simulate ADC data          */
    #define DEBUG_ADC     1          /* stores raw adc conversions in fifo */
    #define DEBUG_LED     0          /* stores tick mark for LED change */
    #define DEBUG_SUMS    0          /* stores the reading for each state */
80  #define DEBUG_RAW      0          /* stores raw ir and raw red      */
    #define DEBUG_HSL     0          /* stores High Speed Link data     */

    /* -----
    Low Speed Link Control
    ----- */

    #define LSL_CHECK_OVERRUN 1      /* check HC11 FIFO            */
    #define LSL_GDD_GROUP_SIZE 6     /* number of gdd packets per lsl_tick */
90  /* -----
    High Speed Link Control
    ----- */

95  #define HSL_RECORD 0          /* records ADC data after ambient subtract */
    #define HSL_PLAYBACK 0        /* uses data from HSL instead of ADC      */
    #define HSL_TEST_PATTERN 0     /* generate rotating 1's pattern          */

    #define IMPULSE 0            /* generate impulses for usr1            */
100 /* -----
    The following constants control the ANC.
    ----- */

105 #define ANC_FIXED_CELL      0      /* fixed cell normalized anc          */

    /* -----
    System Wide Type definitions
    ----- */

110 #define FIXED_CELL      1.0          /* anc_types                        */
    #define ACCORDIAN      2.0          /* anc_types                        */

    /* The following constants describe the size of the following
    structures. */

120 #define MAX_BINS          6
    #define NUM_HR_ALG       7

    #define BIN_RESULTS_SIZE  8
    #define SAT_PARAMS_SIZE  22
125 #define SAT_RESULTS_SIZE  14
    #define HR_PARAMS_SIZE   7
    #define HR_RESULTS_SIZE  8

    #ifndef ASSEMBLY
130 typedef struct {
        INT32      num_points;          /* number of input points          */
        FLOAT32    anc_type;            /* type of anc                      */
        FLOAT32    lambda;              /* nANC parameter                  */
135  #define min_error;                  /* nANC parameter                  */
        INT32      anc_settle_time;     /* nANC parameter                  */
    }

```

```

140  FLOAT32    num_cells;          /* nANC parameter          */
      FLOAT32    peak_limit1;     /* peak picker parameter   */
      FLOAT32    peak_epsilon;    /* peak picker parameters  */
      *bp_coef;    /* pointer to filter coef. */
      INT32      bp_size;         /* number of filter coef.  */
      FLOAT32    search_first;    /* first saturation value. */
      FLOAT32    search_step;     /* saturation step size.   */
      INT32      search_size;     /* number of steps         */
145  INT32      bin_count;        /* number of bins          */
      FLOAT32    cov_limit;       /* covariance limit        */
      FLOAT32    DVHS_init;       /* initial value for DVHS  */
      FLOAT32    DVHS0_init;     /* initial value for first DVHS */
      FLOAT32    DVASt_timer;     /* seconds                 */
150  FLOAT32    DVASt_delta;      /* change for DVASt        */
      FLOAT32    DVVSt_timer;     /* seconds                 */
      FLOAT32    DVVSt_delta;     /* change in DVVSt         */
      FLOAT32    max_sat_delta;   /* max change in sat w/motion */
      INT32      sat_eqn;         /* saturation equation number */
155  FLOAT32    dual_lambda;      /* dual ANC's as VSINCl inputs */
      FLOAT32    correlation;     /* bins OK only if >= corr. */
} SAT_PARAMS;

typedef struct {
160  FLOAT32    ir_rms;           /* signal rms              */
      FLOAT32    rd_rms;         /* signal rms              */
      FLOAT32    cov;           /* covariance               */
      FLOAT32    arterial_sat;   /* resulting arterial sat   */
      FLOAT32    venous_sat;     /* resulting venous sat     */
165  FLOAT32    seed_sat;        /* seed saturation         */
      FLOAT32    num_peaks;      /* peaks in power curve     */
      FLOAT32    correlation;    /* raw dataset correlation  */
} BIN_RESULTS;

170  typedef struct {
      FLOAT32    SpaO2_final;     /* resulting arterial sat   */
      FLOAT32    SpvO2_final;     /* resulting venous sat     */
      FLOAT32    SpaO2_unfil;     /* unfiltered arterial sat  */
      FLOAT32    SpaO2_fill;     /* first stage of filtering */
175  FLOAT32    SpaO2_dual;       /* dual ANC's as input to VSINCl */
      FLOAT32    DVHS;           /* default value high sat   */
      FLOAT32    DVHS0;          /* default value high sat first */
      FLOAT32    DVASt;          /* default value art sat (PR) */
      FLOAT32    DVASt_time;     /* time DVASt was last updated */
180  FLOAT32    DVVSt;           /* default value ven sat (PR) */
      FLOAT32    DVVSt_time;     /* time DVVSt was last updated */
      FLOAT32    art_omega_pr;   /* arterial omega for pulse rate */
      FLOAT32    ven_omega_pr;   /* venous omega for pulse rate */
      FLOAT32    time;           /* time for these results   */
185  BIN_RESULTS bins[MAX_BINS]; /* bin results             */
} SAT_RESULTS;

typedef struct {
190  FLOAT32    resolution;        /* HR resolution in BPM     */
      FLOAT32    raw_sampling_rate; /* raw data sampling rate in Hz */
      FLOAT32    min_hr;         /* minimum hr allowed (BPM) */
      FLOAT32    max_hr;         /* maximum hr allowed (BPM) */
      FLOAT32    nst;           /* noise_spec threshold     */
195  FLOAT32    max_hr_delta;     /* maximum change in HR w/motion */
      INT32      hr_enable;      /* bits control with alg    */
      FLOAT32    hr_lambda;     /* ANC algo. uses this lambda */
} HR_PARAMS;

200  typedef struct {
      FLOAT32    hr[NUM_HR_ALG]; /* (BPM) from each alg      */
      FLOAT32    hr_final;       /* (BPM) after all filtering */
      FLOAT32    BPM_per_bin;    /* BPM per bin of spectrum   */
      INT32      num_bins;       /* number of bins per spectrum */

```

```
205 } HR_RESULTS;  
#endif  
#endif
```

Exhibit G

```

1 ; -----
; Copyright (c) Masimo Corporation (tm) 1992, 1993. All Rights Reserved.
;
5 ; File: stork.gdd
;
; Description: Debug Display Control File
;
; Public Functions:
10 ;
; Private Functions:
;
; Notes:
; Channels 0 to 39 are defined in SPT.H
15 ; Channels 97 to 98 are for status line
;
; History:
;
; RJM 12/10/92 Initial version
20 ; RJM 12/21/92 Add algorithm parameter
; RJM 12/22/92 Add bin results table and plots
; RJM 12/28/92 Add new HR algorithm results
; RJM 12/30/92 Change to one algorithm
; RJM 12/30/92 Add peak_epsilon parameters
25 ; RJM 01/05/93 Add Seed sat to bin data, remove others
; RJM 01/16/93 New SAT results structure
; RJM 01/18/93 Add dvhs_init and dvhs0_init parameters
; RJM 01/18/93 Add hr5 result
; RJM 01/18/93 Add more sat params
30 ; RJM 02/22/93 Real-Time data Plotting
; RJM 02/23/93 Clean up Channel Assignments
;
; -----
;
35 ; Debug1 screen
P 0,2 ; display key = 1
DEBUG SCREEN 1
30,050,025,0375,125,7 ; raw data
34,050,175,0375,275,5 ; analysis data
40 33,050,325,0375,425,5 ; plethy
32,050,475,0375,575,5 ; freq spect
20,775,025,1000,100,5 ; power curve
21,775,150,1000,225,0 ; bin 1-5
22,775,230,1000,305,0
45 23,775,310,1000,385,0
24,775,390,1000,465,0
25,775,470,1000,545,0
31,775,600,1000,725,5 ; ADC Snapshot
00,400,005,0000,000,0 ; system params
50 01,575,005,0000,000,0 ; Sat Results
03,575,225,0000,000,0 ; HR Results
04,575,370,0000,000,0 ; Calib Results
02,010,610,0000,000,0 ; bin results
17,400,490,0014,000,12 ; ADC OOR count
55 98,000,741,0014,000,12 ; 'Working' Text
99,000,741,0015,000,12 ; Version#
97,000,755,0015,000,12 ; file name..
END
;
60 ; Debug2 screen Raw,Spectrum,Plethy, tables
;
P 1,1 ; display key = 2
DEBUG SCREEN 2
30,050,025,425,150,5 ; raw
65 32,050,225,425,330,7 ; freq
33,050,400,425,525,5 ; plethy
00,450,005,000,000,0 ; System Parameters
01,625,005,000,000,0 ; Sat Results

```

```

03,625,225,0000,000,0      ; HR Results
70 04,625,400,0000,000,0      ; Calib Results
    98,050,570,014,000,12      ; 'Working' Text
    99,050,570,015,000,12      ; Version#
    97,050,585,015,000,12      ; file name..
    END
75 ;
    ;      Debug3 screen  power curve,bin graphs
    ;
    P 2,1                      ; display key = 3
    DEBUG SCREEN 3
80 20,050,020,380,130,7      ; power curve
    21,050,180,380,280,7      ; bin 1-5
    22,050,330,380,430,7
    23,450,020,780,130,7
    24,450,180,780,280,7
85 25,450,330,780,430,7
    02,060,460,000,000,0      ; bin results
    98,050,590,014,000,12      ; 'Working' Text
    99,050,590,015,000,12      ; Version#
    97,350,590,015,000,12      ; file name..
90 END
    ;
    ;      Debug4 screen  Raw,Spectrum,Plethy,  tables
    ;
    P 3,0                      ; display key = 4
95  DEBUG SCREEN 4
    30,050,025,270,125,5      ; raw
    32,050,180,270,280,5      ; freq
    33,050,330,270,430,4      ; plethy
    00,300,-15,000,000,0      ; system params
100 01,470,-15,000,000,0      ; sat results
    03,470,195,0000,000,0      ; HR Results
    04,470,320,0000,000,0      ; Calib Results
    98,000,455,014,000,12      ; 'Working' Text
    99,000,455,015,000,12      ; Version#
105 97,000,470,015,000,12      ; file name..
    END
    ;
    ;      Debug5 screen  power curve,bin graphs
    ;
110 P 4,0                      ; display key = 5
    DEBUG SCREEN 5
    20,050,020,295,100,7      ; Master power Curve
    21,050,135,295,215,7      ; bin 1-5
    22,050,250,295,320,7
    23,375,020,620,100,7
    24,375,135,620,215,7
    25,375,250,620,320,7
    02,020,345,000,000,0      ; bin results
    98,020,471,014,000,12      ; 'Working' Text
120 99,020,471,015,000,12      ; Version#
    97,250,471,015,000,12      ; file name..
    END
    ;
    ;      Customer Screen #1
125 ;
    P 5,0                      ; display key = 6
    Customer Screen
    52,000,000,015,000,28      ; stork legend
    77,050,050,450,150,21      ; Real time IR Data (23)
130 78,050,200,450,300,21      ; Real time RD Data
    53,500,080,014,000,16      ; SpaO2 :
    54,500,230,014,000,16      ; Pulse :
    15,500,110,014,000,16      ; SpaO2 value
    16,500,260,014,000,16      ; Pulse value
135 31,500,310,630,400,00      ; ADC Snapshot
    17,500,415,011,000,08      ; ADC OOR count  HGK

```

```

18,500,430,011,000,08      ; Correlation
14,500,445,011,000,08      ; Tick
40,500,460,012,000,08      ; calibrate: 1/0 message
140  END
;
;
;      Customer Screen #2
;
145  P 6,0                  ; display key = 7
Customer Screen
52,000,000,015,000,28      ; stork legend
35,050,050,450,150,21      ; raw analysis portion
33,050,200,450,300,05      ; plethy
150  53,500,080,014,000,16  ; SpaO2 :
54,500,230,014,000,16      ; Pulse :
15,500,110,014,000,16      ; SpaO2 value
16,500,260,014,000,16      ; Pulse value
31,500,310,630,400,00      ; ADC Snapshot
155  17,500,415,011,000,08  ; ADC OOR count  HGK
18,500,430,011,000,08      ; Correlation
14,500,445,011,000,08      ; Tick
40,500,460,012,000,08      ; calibrate: 1/0 message
END
160  ;
;      Front End Screen
;
P 7,1
Front End Screen
165  04,500,010,0000,000,0  ; Calib Results
31,500,200,775,575,7      ; ADC Snapshot
17,050,050,014,000,12      ; ADC OOR count
77,050,200,450,375,23      ; Real time IR Data
78,050,400,450,575,23      ; Real time RD Data
170  40,050,020,011,000,28  ; calibrate: 1/0 message
END
;
; -----
;
175  ;      Channel Definitions
;
; -----
;
;      Usrl snapshot
180  ;
G 30
Raw Data
Point number
Amplitude
185  ;
;      ADC snapshot
;
G 31
ADC SNAPSHOT
190  Point Number
Amplitude
;
;      Frequency Spectrum
;
195  G 32
Spectrum
HR (BPM)
Amplitude
;
200  ;      Master Power Curve
;
G 20
Power Curve
Sat

```

```

205  Amp
    ;
    ;      Bin #1
    ;
    G 21
210  Bin #1
    Sat
    Amp
    ;
    ;      Bin #2
215  ;
    G 22
    Bin #2
    Sat
    Amp
220  ;
    ;      Bin #3
    ;
    G 23
    Bin #3
225  Sat
    Amp
    ;
    ;      Bin #4
    ;
    G 24
    Bin #4
    Sat
    Amp
230  ;
    ;      Bin #5
    ;
    G 25
    Bin #5
    Sat
240  Amp
    ;      Plethysmograph
    ;
    G 33
    Clean Plethysmograph
245  Time
    Amp
    ;      analysis data
    ;
    G 34
250  Analysis Data
    Index
    Amp
    ;      raw analysis data
    ;
255  G 35
    Raw Data (before Masimo)
    Index
    Amp
    ;
260  ;      Info Strings
    ;
    F 98,0
    ---- Working -----
    F 99,0
265  Stork Version %6.3f
    ;
    ;      Algorithm Parameters
    ;
    T 0,1,32
270  ALG 1
    cov_limit,%9.1f
    num_points,%9.0f

```

```

    anc_type,%9.0f
    lambda,%9.4f
275    min_error,%9.4f
    anc_settle,%9.0f
    num_cells,%9.0f
    peak_limit1,%9.2f
    peak_epsilon,%9.2f
280    search_first,%9.2f
    search_step,%9.1f
    search_size,%9.0f
    bin_count,%9.0f
    sampling_rate,%9.0f
285    dvhs_init,%9.1f
    dvhs0_init,%9.1f
    DVAST_timer,%9.0f
    DVAST_delta,%9.1f
    DVVST_timer,%9.0f
290    DVVST_delta,%9.1f
    max_sat_delta,%9.1f
    max_hr_delta,%9.1f
    sat_eqn,%9.0f
    hr_enable,%9.0f
295    Cal Target,%9.4f
    Cal Err Band,%9.4f
    Cal Set. Time,%9.0f
    Cal IR PGA,%9.2f
    Cal RD PGA,%9.2f
300    hr_lambda,%9.4f
    dual_lambda,%9.4f
    Correlation,%9.4f
    ;
    ;      Sat Results
305    ;
    T 1,1,14
    ALG 1
    >SpaO2_final,%9.4f
    >SpvO2_final,%9.4f
310    >SpaO2_unfil,%9.4f
    >SpaO2_fill,%9.4f
    >SpaO2_dual,%9.4f
    DVHS,%9.4f
    DVHS0,%9.4f
315    DVAST,%9.4f
    DVAST time,%9.4f
    DVVST,%9.4f
    DVVST time,%9.4f
    >art_omega_pr,%9.4f
320    >ven_omega_pr,%9.4f
    >time,%9.4f
    ;
    ;      HR Results
    ;
325    T 3,1,8
    Alg 1
    >hr0 (ir),%9.4f
    >hr1 (sig_ref),%9.4f
    >hr2 (deriv),%9.4f
330    >hr3 (ir/n),%9.4f
    >hr4 (sig/n),%9.4f
    >hr5 (spec5),%9.4f
    >hr6 (ANC),%9.4f
    >hr_final,%9.4f
335    ;
    ;      Bin Results
    ;
    T 2,6,8
    Master Bin
340    Bin 1

```

```

Bin 2
Bin 3
Bin 4
Bin 5
345  ir_rms,%9.4f
    rd_rms,%9.4f
    >cov,%9.4f
    >arterial sat,%9.2f
    >venous sat,%9.2f
350  >seed sat,%9.2f
    >num_peaks,%9.0f
    >corr,%9.6f
    ;
    ;      Calib Results
355  ;
    T 4,1,10
    C Results
    ir_trans,%9.4f
    rd_trans,%9.4f
360  desired_ir,%9.4f mA
    desired_rd,%9.4f mA
    actual_ir,%9.4f mA
    actual_rd,%9.4f mA
    ir_gain,%9.2f
365  rd_gain,%9.2f
    ir_delta,%9.4f
    rd_delta,%9.4f
    ;
    ; display strings
370  ;
    M 40,0
    Calibrating
    S 52,1
    Masimo      Stork
375  S 53,0
    SpaO2 :
    S 54,0
    Pulse :
    F 14,0
380  Tick      %12.0f
    F 15,0
    %3.0f %%
    F 16,0
    %3.0f BPM
385  F 17,0
    ADC OOR Count %8.0f
    F 18,0
    Sensor Pos.  %8.4f
    ;
390  ;      Real-Time Plotting
    ;
    R 77,1,400
    Plethysmographics Data (before Masimo)
    Time
395  Amplitude
    R 78,1,400
    Plethysmographics Data (after Masimo)
    Time
    Amplitude
400  X 12,77,0,9
    IR
    X 13,78,0,12
    RD
    ;
405  ; end of items

```

Exhibit H

```

1  /* -----
   Copyright (c) Masimo Corporation (tm) 1992, 1993. All Rights Reserved.

   File: iawa.c

5  Description:
      Industry Algorithm Without Averaging

   Public Functions:
10      init_iawa();
      iawa_sat();

   Private Functions:

15  Notes:

   History:

      RJM      11/05/92      Based on s.c (V1.29)
      RJM      11/20/92      use param and result structures
      RJM      12/08/92      Make compatible with VSINC
      RJM      01/05/93      Compute separate seed for each bin

   ----- */

25  #define MODULE_ID      1015

   #include <masimo.h>      /* platform specification */
   #include <memory.h>      /* special memory allocation */
30  #include <debug.h>      /* assert, grab */
   #include <math.h>
   #include "stork.h"      /* system parameters */
   #include "fir.h"      /* block fir module */
   #include "sateqn.h"      /* saturation equations */
35  #include "iawa.h"      /* self */
   #include "vector.h"      /* cross corr */
   #include "gddcli.h"      /* General Debug Display Client */
   #include "spt.h"

40  #if (PLATFORM == 1)
      extern VOID      /* this BS is to allow UI.C code to store.. */
      presentation(
          FLOAT32      *data,
          INT32      first,
45          INT32      num,
          INT32      code);
   #endif

   VOID
50  init_iawa(void) {
      /* nothing yet */
   }

   VOID
55  compute_log(
      FLOAT32 *in,      /* input data */
      FLOAT32 *out,      /* output data */
      INT32 n) {      /* number of points */

60  /*      This routine computes the natural log of the input signal */
   /*      Because of the filtering in the demodulator, there are times */
   /*      when the resulting signal can go below zero, this log routines */
   /*      limits small values so that the log is gracefully limited. */

65  #define LOG00001 -11.51293

      INT32      i;

```

```

70      BUG1(in); BUG1(out); BUG1(n);

      for (i = 0; i < n; i++) {
          if (in[i] > 0.00001) out[i] = LOGF(in[i]);
          else out[i] = LOG00001;
      }

75  }

VOID
iawa_sat(
80      FLOAT32      *log_ir,          /* input, normalized ir data          */
      FLOAT32      *log_rd,          /* input, normalized rd data          */
      SAT_PARAMS    *sp,              /* input, parameters                  */
      SAT_RESULTS    *sr) {          /* output, results                    */

85      INT32        i;
      INT32        j;
      INT32        b;                /* bin counter                        */
      INT32        bin_size;         /* point in a bin                    */
      INT32        f_size;           /* size of filtered data              */

90      BIN_RESULTS  *r;              /* points to results for each bin     */
      FLOAT32      *f_ir;            /* filtered ir                        */
      FLOAT32      *f_rd;            /* filtered rd                        */
      FLOAT32      *xaxis;           /* for green marker lines             */

95      BUG1(log_ir); BUG1(log_rd); BUG1(sp); BUG1(sr);

      assert(log_ir != NULL);
      assert(log_rd != NULL);
100     assert(sp != NULL);
      assert(sr != NULL);

      f_size = sp->num_points - sp->bp_size;
      bin_size = (f_size - sp->anc_settle_time) / sp->bin_count;
105     assert(f_size > 0);

      mark_mem();
      f_ir = get_mem(f_size);
      f_rd = get_mem(f_size);
110     xaxis = get_mem(1000);

      /* band-pass filter IR and RD signals */

      blkfir(log_ir, sp->num_points, sp->bp_coef, sp->bp_size, f_ir);
115     blkfir(log_rd, sp->num_points, sp->bp_coef, sp->bp_size, f_rd);

      /* output analysis data ----- */

      b=0; /* build a dataset that demarks the bins divisions */
120     assert(f_size-sp->anc_settle_time < 1000);
      for(i=0; i < f_size - sp->anc_settle_time; i++)
      {
          xaxis[i]=(FLOAT32) (b-5000);
          if(!(i%bin_size)) b=10000-b;
125     }

      j = sp->anc_settle_time- 1;
      b = f_size - j;
      GDDCLI_Chart(SPT_AnalysisData,3,b, 0.0, 1.0);
130     GDDCLI_Data (SPT_AnalysisData,b,&f_rd[j]);
      GDDCLI_Data (SPT_AnalysisData,b,&f_ir[j]);
      GDDCLI_Data (SPT_AnalysisData,b,xaxis);

      #if (PLATFORM == 1)
135     presentation(f_rd,j,b,0);
      #endif

```

```

/* calculate correlations ----- */
140 sr->bins[0].correlation = VECTOR_CrossCorr0(&f_ir[j], &f_rd[j], f_size-j);
for (b = 1; b <= sp->bin_count; b++)
{
    r = &(sr->bins[b]);
145 r->correlation = VECTOR_CrossCorr0(&f_ir[j], &f_rd[j], bin_size);
    j+= bin_size;
}

/* Square both signals */
150 for (i = 0; i < f_size; i++)
{
    f_ir[i] *= f_ir[i];
    f_rd[i] *= f_rd[i];
155 }

/* Integrate using binning */

160 sr->bins[0].ir_rms = 0.0;
sr->bins[0].rd_rms = 0.0;
j = sp->anc_settle_time - 1;

for (b = 1; b <= sp->bin_count; b++)
{
165 r = &(sr->bins[b]);

    r->ir_rms = 0.0;
    r->rd_rms = 0.0;

170 for (i = 0; i < bin_size; i++)
    {
        r->ir_rms += f_ir[j];
        r->rd_rms += f_rd[j];
        j++;
175 }

    /* update master bin */

    sr->bins[0].ir_rms += r->ir_rms;
180 sr->bins[0].rd_rms += r->rd_rms;

    r->ir_rms = SQRTF(r->ir_rms / bin_size);
    r->rd_rms = SQRTF(r->rd_rms / bin_size);
    r->seed_sat = calc_sat(r->rd_rms / r->ir_rms, sp->sat_eqn);
185 }

/* compute master bin results */

190 r = &(sr->bins[0]);
r->ir_rms = SQRTF(r->ir_rms / f_size); /* was bin_size HGK */
r->rd_rms = SQRTF(r->rd_rms / f_size);
r->seed_sat = calc_sat(r->rd_rms / r->ir_rms, sp->sat_eqn);
free_mem();
195 }

```

Exhibit I

```

1  /* -----
   Copyright (c) Masimo Corporation (tm) 1992. All Rights Reserved.

   File: satfil.c

5  Description: SpaO2 output filtering algorithms

   Public Functions:
       reset_satfil      Resets all filters
10      sat_filter        runs all sat filters

   Private Functions:
       max_diff_alg

15  Notes:

   History:

       RJM      10/09/92      Initial version
20      RJM      10/12/92      Allow multiple filtering streams
       RJM      11/06/92      Bring up to coding standards
       RJM      01/16/93      Merge binning algorithms, DVHS filters, etc
       RJM      01/18/93      If COV is low, and 1 peak don't report SvO2
       RJM      01/26/93      Limit final SpaO2 by highest seed of all bins
25      RJM      01/26/93      Compare max-diff values to SpaO2_final

   ----- */

30  #define MODULE_ID      1004

   #include <masimo.h>      /* platform description file      */
   #include "stork.h"      /* system parameters              */
   #include "satfil.h"      /* self                           */
   #include "sateqn.h"      /* calc_omega                     */
35  #include <memory.h>      /* Special memory routines        */
   #include <debug.h>      /* assert, grab                   */
   #include <vector.h>      /* vector functions               */
   #include <math.h>      /*

40  VOID
reset_satfil(
    SAT_PARAMS *sp,          /* input, saturation parameters    */
    SAT_RESULTS *sr) {      /* inout, saturation results      */

45      BUG1(sp); BUG1(sr);

      assert(sp != NULL);
      assert(sr != NULL);

50      sr->DVHS = sp->DVHS_init;
      sr->DVHS0 = sp->DVHS0_init;
      sr->SpaO2_final = 0.0;
      sr->SpaO2_unfil = 0.0;
      sr->SpaO2_fill = 0.0;
55      sr->SpvO2_final = 0.0;
      sr->SpaO2_dual = 0.0;
      sr->DVASt = 0.0;
      sr->DVVSt = 0.0;
      sr->DVASt_time = -sp->DVASt_timer - 1;
60      sr->DVVSt_time = -sp->DVVSt_timer - 1;

}

65  /* -----

   Name:      max_diff_alg

```

Abstract: This routine finds the two bins with the largest difference between arterial and venous sat and of these two returns the saturation which is closest to the previous sat. For more detail see Stork design note 27.

Algorithm:

1. Select the bins that have a good arterial and venous saturation and order these bins based on the difference between the venous and the arterial saturation.
2. Select the two bins with the greatest difference. If there are no bins resulting from step 1.
3. Of these two bins select the one closest to the previous SpaO2_final value.
4. If there is no previous value, then return the largest value.

----- */

```
static FLOAT32
max_diff_alg(
    SAT_PARAMS *sp,          /* input, saturation parameters */
    SAT_RESULTS *sr) {      /* inout, saturation results */

    INT32 i;
    BIN_RESULTS *b;          /* general bin ptr */
    INT32 *good_bins;        /* array of good bin numbers */
    FLOAT32 *diffs;          /* array of differences */
    INT32 num_good_bins;     /* number of good bins */
    FLOAT32 sat;             /* return value */
    FLOAT32 sat1;            /* arterial sat of bin with largest diff */
    FLOAT32 sat2;            /* sat of bin with 2nd largest diff. */

    BUG1(sp); BUG1(sr);

    assert(sp != NULL);
    assert(sr != NULL);

    mark_mem();
    good_bins = get_mem(sp->bin_count);
    diffs = get_mem(sp->bin_count);

    /* select bins that have good arterial and venous sat */
    /* R154 : bin must also have good correlation HGK */

    num_good_bins = 0;
    for (i = 1; i <= sp->bin_count; i++) {
        b = &(sr->bins[i]);
        if (b->arterial_sat > 0 && b->venous_sat > 0 &&
            b->correlation >= sp->correlation)
        {
            good_bins[num_good_bins] = i;
            diffs[num_good_bins] = b->arterial_sat - b->venous_sat;
            num_good_bins++;
        }
    }

    /* Find Two bins with largest difference */

    switch (num_good_bins) {

        case 0:
            sat = sr->SpaO2_unfil;
            break;

        case 1:
```

```

        i = max_index(diffs, num_good_bins);
        sat = sr->bins[good_bins[i]].arterial_sat;
        break;
140
    default:
        /* find bin with largest diff */

        i = max_index(diffs, num_good_bins);
145        sat1 = sr->bins[good_bins[i]].arterial_sat;

        /* find bin with second largest diff */

        diffs[i] = 0.0;
        i = max_index(diffs, num_good_bins);
150        sat2 = sr->bins[good_bins[i]].arterial_sat;

        /* If there is no previous SpaO2 choose highest sat */

155        if (sr->SpaO2_final == 0.0) {
            sat = sat1;
            if (sat2 > sat1) sat = sat2;
        } else {

160            /* Otherwise, choose one closest to SpaO2_final */

            sat = sat1;
            if (FABSF(sat1 - sr->SpaO2_final) >
                FABSF(sat2 - sr->SpaO2_final)) sat = sat2;
165        }
        break;

    }

170    free_mem();
    return(sat);
}

/* -----
175
    Name:          sat_filter

    Abstract:      This routine takes the master power curve results and the
                    five bins and computes an overall arterial and venous
180                    saturation based on the following algorithm. For more
                    detail see Stork design note 27.

    Algorithm:

185
    ----- */

#define MAX_SPAO2      99.0    /* max sat for final filter output    */
#define MIN_SPAO2      40.0    /* min sat for final filter output    */
190
VOID
sat_filter(
    SAT_PARAMS *sp,           /* input, saturation parameters      */
    SAT_RESULTS *sr) {       /* inout, saturation results         */
195
    BIN_RESULTS *mb;         /* master bin ptr                    */
    FLOAT32      delta;
    INT32         i;
    FLOAT32      max_seed;    /* maximum seed from all bins        */
200

    BUG1(sp); BUG1(sr);

    assert(sp != NULL);

```

```

205     assert(sr != NULL);

    /* If master bin cov is less than limit then use its sat value */
    /* otherwise, use the sat of the max_diff_alg of the bins. */

210     mb = &(sr->bins[0]);
    if (mb->cov < sp->cov_limit) {
        sr->SpaO2_unfil = mb->arterial_sat;

        /* Update DVHS */

215         if (mb->arterial_sat > sr->DVHS0) {
            sr->DVHS = mb->arterial_sat;
            sr->DVHS0 = sr->DVHS;
        }

220         /* DVAST algorithm */

        if (mb->arterial_sat > 0.0 && mb->arterial_sat < sr->DVHS) {
            sr->DVAST = mb->arterial_sat;
225             sr->DVAST_time = sr->time;
        }

    } else {
        sr->SpaO2_unfil = max_diff_alg(sp, sr);

230         /* DVVST algorithm */

        if (mb->venous_sat > 0.0 && mb->num_peaks > 1) {
            sr->DVVST = mb->venous_sat;
235             sr->DVVST_time = sr->time;
        }

    }

    /* first stage of filtering (DVHS) */

240     sr->SpaO2_fill = sr->SpaO2_unfil;
    if (sr->DVHS < sr->SpaO2_fill) sr->SpaO2_fill = sr->DVHS;

    /* second stage of filtering */
    /* If there is motion and a previous value, limit the change */
    /* to max_sat_delta. */

    if (mb->cov > sp->cov_limit && sr->SpaO2_final > 0.0) {
        delta = sr->SpaO2_fill - sr->SpaO2_final;
250         if (delta > sp->max_sat_delta) delta = sp->max_sat_delta;
        if (delta < -sp->max_sat_delta) delta = -sp->max_sat_delta;
        sr->SpaO2_final += delta;
    } else {
        sr->SpaO2_final = sr->SpaO2_fill;
255     }

    /* Find largest seed from all bins */

260     max_seed = mb->seed_sat;
    for (i = 1; i <= sp->bin_count; i++) {
        if (sr->bins[i].seed_sat > max_seed) max_seed = sr->bins[i].seed_sat;
    }

265     if (sr->SpaO2_final < max_seed) sr->SpaO2_final = max_seed;
    if (sr->SpaO2_final > MAX_SPAO2) sr->SpaO2_final = MAX_SPAO2;
    if (sr->SpaO2_final < MIN_SPAO2) sr->SpaO2_final = MIN_SPAO2;

    /* Venous logic */

270     if (mb->cov < sp->cov_limit && mb->num_peaks < 2) {
        sr->SpvO2_final = 0.0;
    }

```

```

    } else {
        sr->SpvO2_final = mb->venous_sat;
275     }

    /* Compute arterial and venous sat for use by Pulse Rate algorithms */

    sr->art_omega_pr = 0.0;
280     if (sr->time - sr->DVASt_time < sp->DVASt_timer) {
        if (FABSF(sr->DVASt - sr->SpaO2_fill) < sp->DVASt_delta) {
            sr->DVASt_time = sr->time;
        }
        sr->art_omega_pr = calc_omega(sr->DVASt, sp->sat_eqn);
285     } else {
        if (sr->SpaO2_fill > 0.0) {
            sr->art_omega_pr = calc_omega(sr->SpaO2_fill, sp->sat_eqn);
        }
    }
290

    sr->ven_omega_pr = 0.0;

    if (sr->time - sr->DVVST_time < sp->DVVST_timer) {
295         if (FABSF(sr->DVVST - mb->venous_sat) < sp->DVVST_delta) {
            sr->DVVST_time = sr->time;
        }
        sr->ven_omega_pr = calc_omega(sr->DVVST, sp->sat_eqn);
    } else {
300         if (mb->venous_sat > 0.0 &&
            mb->cov > sp->cov_limit && mb->num_peaks > 1) {
            sr->ven_omega_pr = calc_omega(mb->venous_sat, sp->sat_eqn);
        }
    }
305

} /* satfil */

310 #ifdef STANDALONE

#include <stdlib.h>
#include <stdio.h>

315 VOID main(void) {

}

320 #endif

```

Exhibit K

```

1  /* -----
   Copyright (c) Masimo Corporation (tm) 1992, 1993. All Rights Reserved.

5  File: ui.c

Description:
This is the STORK software user interface module. This platform
independant user interface provides the ability to start and stop
the oximeter, change system parameters, and control the input data.
10 There are minor differences in these features listed below:

Feature                PC Platform                DSP Platform
-----
15 Start/Stop           Yes                      Yes
Change system param    Yes                      Yes
LED calibration        No                       Yes
PAN through data       Yes                      No
Rerun same data        Yes                      Yes
Standard Mode          Yes                      Yes

20 Public Functions:
init_ui                Standard initialization
user_interface          Start the user interface

25 Private Functions:
store_raw              send raw data via LSL
store_param            send parameters via LSL
store_results          send results via LSL
single_step            Run a single snapshot of data
30 test_pattern         send a test pattern via LSL

Notes:

History:
35 RJM      12/09/92      Initial Version based on main.c
RJM      12/28/92      New HR module vsi_hr
RJM      12/30/92      New default values for parameters
RJM      12/30/92      Add peak_epsilon parameter
RJM      01/18/93      Add dvhs_init and dvhs0_init parameters
40 RJM      01/18/93      Add hrfil module
RJM      01/18/93      Keep track of time
RJM      01/25/93      Set default for DVHS0 to 95
RJM      01/25/93      Make SAT eqn parameter
RJM      01/25/93      hr_enable param
45 RJM      02/02/93      Add command for DSP Platform version
RJM      02/11/93      Validate data after get_snapshot
RJM      02/12/93      New default values for coefs
RJM      02/19/93      Integrate file list feature from HK.
RJM      02/23/93      Add Calibration parameters
50 RJM      02/24/93      Improve ADC Snapshot
HGK      03/08/93      move data_ptr code to usr1.c
----- */

55 #define MODULE_ID      1024

#include <masimo.h>      /* Platform description      */
#include <pckeybrd.h>    /* key definitions          */
#include <memory.h>      /* memory manager           */
#include <debug.h>       /* virtual platform debugger */
60 #include <math.h>

#include "sateqn.h"      /* saturation equations     */
#include "stork.h"       /* System parameters        */
#include "adc.h"         /* Demodulation module      */
#include "usr1.h"        /* Filtering and Decimation */
65 #include "calib.h"     /* LED calibration          */
#include "vsinc.h"       /* VSI noise cancellation module */
#include "satfil.h"      /* Saturation Filters       */
#include "hrfil.h"       /* Pulse Rate Filters       */

```

```

#include "lsl.h"          /* low speed link module          */
70  #include "iawa.h"      /* Industry algorithm without averaging */
#include "post.h"         /* post band pass filter          */
#include "vsihr.h"        /* VSI Heart Rate Module          */
#include "vsipl.h"        /* VSINC Clean photoplethysmograph */
#include "gddcli.h"       /* General Debug Display Client    */
75  #include "spt.h"       /* Stork Packet Types             */
#include "vsidual.h"      /* dual ANC sat version           */
#include "ui.h"           /* Self                           */

#if (PLATFORM == 1)
80  #include <stdio.h>
static BOOL presentation_flag;
#endif

typedef struct {
85  FLOAT32    min_val;    /* minimum value          */
    FLOAT32    max_val;    /* maximum value          */
    FLOAT32    inc_val;    /* amount to change       */
    FLOAT32    def_val;    /* default value          */
} PARAM_DESC;

90

typedef struct {
    SAT_PARAMS    *vp;    /* pointer to vsinc params          */
    SAT_RESULTS    *vr;    /* pointer to vsinc results         */
95  FLOAT32    *power;    /* pointer to power curve          */
    HR_PARAMS    *vhp;    /* pointer to Heart Rate params     */
    HR_RESULTS    *vhr;    /* pointer to Heart Rate results    */
    FLOAT32    *fspec;    /* frequency spectrum              */
    FLOAT32    *pleth;    /* clean plethysmograph            */
100  CALIB_PARAMS    *cp;    /* pointer to calibration params    */
    CALIB_RESULTS    *cr;    /* pointer to calibration results   */
} ALGORITHM_SPEC;

105  /*      The following constants control the user adjustable      */
/*      parameters.                                                  */

#define MAX_ALGORITHMS    1
110  #define NUM_PARAMS      32

#define P_cov_limit        0
#define P_num_points      1
#define P_anc_type        2
115  #define P_lambda        3
#define P_min_error        4
#define P_anc_settle_time  5
#define P_num_cells        6
#define P_peak_limit1      7
120  #define P_peak_epsilon  8
#define P_search_first     9
#define P_search_step     10
#define P_search_size     11
#define P_bin_count       12
125  #define P_sampling_rate 13
#define P_dvhs_init       14
#define P_dvhs0_init      15
#define P_DVAST_timer     16
#define P_DVAST_delta     17
130  #define P_DVVST_timer   18
#define P_DVVST_delta     19
#define P_max_sat_delta   20
#define P_max_hr_delta    21
#define P_sat_eqn         22
135  #define P_hr_enable     23
#define P_cp_target       24

```

```

#define P_cp_error_band      25
#define P_cp_settle_time     26
#define P_cp_ir_gain         27
140 #define P_cp_rd_gain       28
#define P_hr_lambda          29
#define P_dual_lambda        30
#define P_correlation        31

145 static PARAM_DESC params[NUM_PARAMS] = {
    /*      min      max      inc      def */
    /* cov_limit      0 */ { 0.0, 200.0, 2.0, 40.0 },
    /* num_points     1 */ { 100.0, 1000.0, 10.0, 650.0 },
    /* anc_type       2 */ { 1.0, 2.0, 1.0, 1.0 },
150  /* lambda         3 */ { 0.0, 1.0, 0.0001, 0.75 },
    /* min_error      4 */ { 0.0, 1.0, 0.0001, 0.0005 },
    /* anc_settle_time 5 */ { 0.0, 1000.0, 1.0, 149.0 },
    /* num_cells      6 */ { 1.0, 50.0, 1.0, 12.0 },
    /* peak_limit1    7 */ { 0.0, 1.0, 0.01, 0.02 },
155  /* peak_epsilon   8 */ { 0.0, 1.0, 0.01, 0.03 },
    /* search_first   9 */ { 0.0, 120.0, 0.25, 104.0 },
    /* search_step   10 */ { 0.0, 5.0, 0.10, 0.5 },
    /* search_size   11 */ { 0.0, 300.0, 5.0, 195.0 },
    /* bin_count     12 */ { 1.0, 5.0, 1.0, 5.0 },
160  /* sampling rate  13 */ { 10.0, 200.0, 1.0, 62.5 },
    /* dvhs_init     14 */ { 40.0, 105.0, 0.1, 99.0 },
    /* dvhs0_init    15 */ { 40.0, 105.0, 0.1, 95.0 },
    /* DVASt_timer   16 */ { 0.0, 100.0, 1.0, 10.0 },
    /* DVASt_delta   17 */ { 0.0, 100.0, 0.1, 2.0 },
165  /* DVVSt_timer   18 */ { 0.0, 100.0, 1.0, 10.0 },
    /* DVVSt_delta   19 */ { 0.0, 100.0, 0.1, 4.0 },
    /* max_sat_delta 20 */ { 0.0, 20.0, 0.1, 2.0 },
    /* max_hr_delta  21 */ { 0.0, 40.0, 0.1, 4.0 },
    /* sat_eqn       22 */ { 0.0, 1.0, 1.0, 1.0 },
170  /* hr_enable     23 */ { 0.0, 255.0, 1.0, 251.0 },
    /* cp_target     24 */ { 0.0, 1.0, 0.05, 0.25 },
    /* cp_error_band 25 */ { 0.0, 0.5, 0.01, 0.10 },
    /* cp_settle_time 26 */ { 0.0, 300.0, 2.00, 75.0 },
    /* cp_ir_gain     27 */ { 0.0, 4096.0, 0.1, 2.0 },
175  /* cp_rd_gain     28 */ { 0.0, 4096.0, 0.1, 2.0 },
    /* hr_lambda      29 */ { 0.1, 1.0, 0.0001, 0.9999 },
    /* dual_lambda    30 */ { 0.1, 1.0, 0.0001, 0.9999 },
    /* correlation    31 */ { 0.1, 1.0, 0.0001, 0.9999 },
};

180 static FLOAT32 param_val[MAX_ALGORITHMS][NUM_PARAMS];

VOID
init_ui(void) {
185     /* nothing yet */
}

/* =====
store raw,plethy data to ASCII files to prepare for presentations
===== */
190 #if (PLATFORM == 1)

VOID
presentation(
195     FLOAT32 *data,
     INT32 first,
     INT32 num,
     INT32 code)
{
200     static CHAR filename[65];
     CHAR str[65];
     FILE *fp;
     INT32 i;

```

```

205     if(!presentation_flag) return;

        switch(code) {
            case 0 : printf("Enter B/A filename (.B,.A will be added) : ");
                      scanf("%s",filename);
210                      sprintf(str,"%s.B",filename);
                      goto part2;
            case 1 : sprintf(str,"%s.A",filename);
part2:
                      if((fp=fopen(str,"w+"))==NULL) break;
215                      for(i=0;i<num;i++) {
                          fprintf(fp,"%f\n",data[first+i]);
                      }
                      fclose(fp);
                      break;
220        }
    }
#endif

225 static VOID
store_raw(
    SAT_PARAMS *sp,          /* input, parameters          */
    FLOAT32 *ir,             /* input, ir datav           */
    FLOAT32 *rd,             /* input, red data           */
230    INT32 size,             /* input, number of points   */
    INT32 time_stamp) {     /* input, starting point number */

    INT32 i,j,k;
    FLOAT32 *t;
235
    BUG1(sp); BUG1(ir); BUG1(rd); BUG1(size); BUG1(time_stamp);

    #if (PLATFORM == 1)

240        mark_mem();
        t = get_mem(size);
        assert(t != NULL);
        j = sp->bp_size/2 + sp->anc_settle_time;    /* 1st pt */
        k = size - sp->bp_size - sp->anc_settle_time; /* #usable pts */
245

        for(i=0;i<j ;i++) t[i] = -10;
        for( ;i<j+k ;i++) t[i] = 10;
        for( ;i<size;i++) t[i] = -10;
250        GDDCLI_Chart(SPT_Usr1SnapShot, 3, size, (FLOAT32)time_stamp, 1.0);
        GDDCLI_Data(SPT_Usr1SnapShot, size, ir);
        GDDCLI_Data(SPT_Usr1SnapShot, size, rd);
        GDDCLI_Data(SPT_Usr1SnapShot, size, t );

255        GDDCLI_Chart(SPT_RawAnalysisData, 2,k, (FLOAT32)time_stamp, 1.0);
        GDDCLI_Data (SPT_RawAnalysisData, k, &ir[j]);
        GDDCLI_Data (SPT_RawAnalysisData, k, &rd[j]);

        free_mem();
260    #endif

    }

    static VOID
265    store_parameters(
        INT32 num_algorithms) {

        INT32 i;

270        BUG1(num_algorithms);

        for (i = 0; i < num_algorithms; i++) {

```

```

        GDDCLI_Data(SPT_SystemParam, NUM_PARAMS, param_val[i]);
    }
275 }

static VOID
store_results(
280     ALGORITHM_SPEC      *alg,
        INT32              num_algorithms) {

        SAT_PARAMS        *vp;
        SAT_RESULTS        *vr;
285     HR_PARAMS          *vhp;
        HR_RESULTS         *vhr;
        FLOAT32            *power;
        FLOAT32            *fspec;          /* Frequency SPECTrum          */
        FLOAT32            *pleth;         /* clean pleth.                */
290     INT32               i;
        INT32               j;
        FLOAT32            temp;

        BUG1(alg); BUG1(num_algorithms);

295     mentry("store_results");

        /* frequency spectrum */

300     i = 0;
        vp = alg[i].vp;
        vr = alg[i].vr;
        power = alg[i].power;
        vhp = alg[i].vhp;
305     vhr = alg[i].vhr;
        fspec = alg[i].fspec;
        pleth = alg[i].pleth;

        /* frequency spectrum */
310     GDDCLI_Chart(SPT_FreqSpec, 1, vhr->num_bins, vhp->min_hr,
                    vhr->BPM_per_bin);
        GDDCLI_Data(SPT_FreqSpec, vhr->num_bins, alg[i].fspec);

315     /* power curve */
        GDDCLI_Chart(SPT_MasterPowerCurve, num_algorithms, vp->search_size,
                    vp->search_first, -(vp->search_step));

320     for (i = 0; i < num_algorithms; i++) {
        GDDCLI_Data(SPT_MasterPowerCurve, vp->search_size, alg[i].power);
    }

    for (j = 0; j < vp->bin_count; j++) {
325     GDDCLI_Chart(SPT_PowerCurve1 + j, num_algorithms, vp->search_size,
                    vp->search_first, -(vp->search_step));

        for (i = 0; i < num_algorithms; i++) {
330     GDDCLI_Data(SPT_PowerCurve1 + j, vp->search_size,
                    &(alg[i].power[(j+1)*vp->search_size]));
        }
    }

335     /* plethysmographic signal */
        GDDCLI_Chart(SPT_CleanPleth, num_algorithms, pleth_size(vp), 0.0, 1.0);
        for (i = 0; i < num_algorithms; i++) {
340     GDDCLI_Data(SPT_CleanPleth, pleth_size(vp), alg[i].pleth);
        }

```

```

    #if (PLATFORM == 1)
        presentation(alg[0].pleth, 0, pleth_size(vp), 1);
    #endif
345     /* store tabular results */

    for (i = 0; i < num_algorithms; i++) {
        GDDCLI_Data(SPT_SatResults, SAT_RESULTS_SIZE, (FLOAT32 *)alg[i].vr);
350     GDDCLI_Data(SPT_HRResults, HR_RESULTS_SIZE, (FLOAT32 *)alg[i].vhr);
    }

    for (i = 0; i <= alg[0].vp->bin_count; i++) {
        GDDCLI_Data(SPT_BinResults, BIN_RESULTS_SIZE,
355     (FLOAT32 *)&(alg[0].vr->bins[i]));
    }

    /* master bins' correlation sent separately (times 100)----- */
    temp = alg[0].vr->bins[0].correlation * 100.0;
360     GDDCLI_Data(SPT_correlation, 1, &temp);

    mexit("store_results");
}

365 static INT32
compute_snapshot_size(
    ALGORITHM_SPEC      *alg,
    INT32                num_algorithms) {

370     INT32      i;
    INT32      size;

    BUG1(alg); BUG1(num_algorithms);

375     /* find largest snapshot size */

    size = 0;
    for (i = 0; i < num_algorithms; i++) {
        if (alg[i].vp->num_points > size) size = alg[i].vp->num_points;
380     }
    return(size);
}

static VOID
385 new_patient(
    ALGORITHM_SPEC      *alg,
    INT32                num_algorithms) {

    INT32      i;

390     BUG1(alg); BUG1(num_algorithms);

    for (i = 0; i < num_algorithms; i++) {
        reset_satfil(alg[i].vp, alg[i].vr);
395     reset_hrfil(alg[i].vhp, alg[i].vhr);
    }
}

400 static VOID
single_step(
    ALGORITHM_SPEC      *alg,          /* input, algorithm structure */
    INT32                num_algorithms,
    INT32                raw_snapshot_size) {

405     INT32      i;          /* algorithm */
    FLOAT32      *ir;        /* raw ir data */

```

```

410  FLOAT32      *rd;          /* raw rd data          */
      FLOAT32      *log_ir;     /* log of ir data       */
      FLOAT32      *log_rd;     /* log of rd data       */
      FLOAT32      *adc_snap;   /* snapshot of adc data */

      SAT_PARAMS   *vp;
415  SAT_RESULTS   *vr;
      HR_PARAMS    *vhp;
      HR_RESULTS   *vhr;
      FLOAT32      *power;
      FLOAT32      *fspec;      /* Frequency SPECTrum   */
420  FLOAT32      *pleth;       /* clean pleth.         */
      INT32        time;

      BUG1(alg);  BUG1(num_algorithms);  BUG1(raw_snapshot_size);

425  mentry("single_step");
      mark_mem();

      /* Allocate data required for current algorithms */

430  ir           = get_mem(raw_snapshot_size);
      rd          = get_mem(raw_snapshot_size);
      log_ir      = get_mem(raw_snapshot_size);
      log_rd      = get_mem(raw_snapshot_size);
435  adc_snap     = get_mem(ADC_SNAPSHOT_SIZE);

      ADC_GetSnapshot(adc_snap);
      adc_snap[0] = -1.0;
      adc_snap[1] = 1.0;

440  GDDCLI_Chart(SPT_ADCSnapshot, 1, ADC_SNAPSHOT_SIZE, 0.0, 1.0);
      GDDCLI_Data(SPT_ADCSnapshot, ADC_SNAPSHOT_SIZE, adc_snap);

      GDDCLI_SendPacket(SPT_ADC_OOR, (FLOAT32)ADC_GetOutOfRangeCount());

445  time = get_snapshot(ir, rd, raw_snapshot_size);

      /* validate this snapshot of data */

      for (i = 0; i < raw_snapshot_size; i++) {
450      if (ir[i] > 2.0) assert(0);
          if (ir[i] < -2.0) assert(0);
          if (rd[i] > 2.0) assert(0);
          if (rd[i] < -2.0) assert(0);
      }

455  store_raw(alg[0].vp, ir, rd, raw_snapshot_size, time);

      compute_log(ir, log_ir, raw_snapshot_size);
      compute_log(rd, log_rd, raw_snapshot_size);

460  for (i = 0; i < num_algorithms; i++) {

          /* setup pointers for current algorithm */

465  vp = alg[i].vp;
      vr = alg[i].vr;
      vhp = alg[i].vhp;
      vhr = alg[i].vhr;

470  /* allocate variable sized results arrays */

          alg[i].power = get_mem(vp->search_size * (vp->bin_count + 1));
          alg[i].fspec = get_mem(vsi_spectral_size(vhp));
          alg[i].pleth = get_mem(pleth_size(vp));

475  power = alg[i].power;

```

```

        fspec = alg[i].fspec;
        pleth = alg[i].pleth;

480    /* time stamp data */

        vr->time = (FLOAT32)time / vhp->raw_sampling_rate;

        iawa_sat(log_ir, log_rd, vp, vr);

485    vsinc(ir, log_ir, log_rd, vp, vr, power);
        sat_filter(vp, vr);

    /*
490    vsi_hr( log_ir, log_rd, vp, vr, vhp, vhr, fspec); */
        vsi_hr(ir, log_ir, log_rd, vp, vr, vhp, vhr, fspec);

        hr_filter(vhp, vhr, vp, vr);

    /*
495    vsinc_pl(ir, log_ir, log_rd, vr->art_omega_pr, vp, pleth); */
        vsinc_pl(ir, log_ir, log_rd,
                vr->art_omega_pr,
                vhp->hr_lambda, /* HR's lambda for plethy. */
                vp, pleth);

500    vr->SpaO2_dual = 0.0;

    /*
        VSIDUAL_sat(vr->art_omega_pr,
                vp->dual_lambda,
                vp,
505    ir, rd, log_ir, log_rd,
                &vr->SpaO2_dual);
    */

        GDDCLI_SendPacket(SPT_time_tick, (FLOAT32)time);
        GDDCLI_SendPacket(SPT_sat_final, vr->SpaO2_final);
510    GDDCLI_SendPacket(SPT_hr_final, vhr->hr_final);

    } /* for */

        store_parameters(num_algorithms);
515    store_results(alg, num_algorithms);

        GDDCLI_NewLine();
        GDDCLI_FlushBuffer(); /* send data */

520    free_mem();
        mexit("single_step");
    }

static VOID
525 test_pattern(void) {

    /* send a test pattern to verify graphing stuff */

}

530 static VOID
set_alg(
    ALGORITHM_SPEC    *alg,
    INT32              num_alg) {

535    SAT_PARAMS        *vp;
    HR_PARAMS          *vhp;
    CALIB_PARAMS       *cp;

540    INT32              i;

    BUG1(alg); BUG1(num_alg);

    for (i = 0; i < num_alg; i++) {

```

```

545      vp = alg[i].vp;

      vp->cov_limit      = param_val[i][P_cov_limit];
      vp->num_points     = (INT32)param_val[i][P_num_points];
550      vp->anc_type      = (INT32)param_val[i][P_anc_type];
      vp->lambda         = param_val[i][P_lambda];
      vp->min_error      = param_val[i][P_min_error];
      vp->anc_settle_time = (INT32)param_val[i][P_anc_settle_time];
      vp->num_cells      = (INT32)param_val[i][P_num_cells];
555      vp->peak_limit1   = param_val[i][P_peak_limit1];
      vp->peak_epsilon   = param_val[i][P_peak_epsilon];
      vp->bp_coef        = post_coef;
      vp->bp_size        = post_size;
      vp->search_first   = param_val[i][P_search_first];
560      vp->search_step   = param_val[i][P_search_step];
      vp->search_size    = (INT32)param_val[i][P_search_size];
      vp->bin_count      = (INT32)param_val[i][P_bin_count];
      vp->DVHS_init      = param_val[i][P_dvhs_init];
      vp->DVHS0_init     = param_val[i][P_dvhs0_init];
565      vp->DVASt_timer   = param_val[i][P_DVASt_timer];
      vp->DVASt_delta    = param_val[i][P_DVASt_delta];
      vp->DVASt_timer    = param_val[i][P_DVASt_timer];
      vp->DVASt_delta    = param_val[i][P_DVASt_delta];
      vp->max_sat_delta  = param_val[i][P_max_sat_delta];
570      vp->sat_eqn       = (INT32)param_val[i][P_sat_eqn];

      vhp = alg[i].vhp;
      vhp->resolution    = 1.0;
      vhp->raw_sampling_rate = (INT32)param_val[i][P_sampling_rate];
575      vhp->min_hr       = 25.0;      /* bpm */
      vhp->max_hr        = 225.0;    /* bpm */
      vhp->nst           = 0.04;     /* threshold */
      vhp->max_hr_delta  = param_val[i][P_max_hr_delta];
      vhp->hr_enable     = (INT32)param_val[i][P_hr_enable];
580      vhp->hr_lambda    = param_val[i][P_hr_lambda];
      vp->dual_lambda    = param_val[i][P_dual_lambda];
      vp->correlation     = param_val[i][P_correlation];

585      cp = alg[i].cp;
      cp->rd_min_mA      = 20.0;      /* mA */
      cp->rd_max_mA      = 50.0;      /* mA */
      cp->ir_min_mA      = 5.0;       /* mA */
      cp->ir_max_mA      = 50.0;      /* mA */
590      cp->target        = param_val[i][P_cp_target];
      cp->error_band     = param_val[i][P_cp_error_band];
      cp->ir_drive_1     = 5.0;       /* mA */
      cp->rd_drive_1     = 20.0;      /* mA */
      cp->ir_drive_2     = 10.0;      /* mA */
595      cp->rd_drive_2     = 25.0;      /* mA */
      cp->settle_time    = (INT32)param_val[i][P_cp_settle_time];
      cp->ir_gain        = param_val[i][P_cp_ir_gain];
      cp->rd_gain        = param_val[i][P_cp_rd_gain];
      cp->mu             = 0.1;

600  }
}

#define DATA_STEP 57      /* amount to scroll raw data */
605  VOID
  user_interface(
    INT32      num_algorithms) {

610    ALGORITHM_SPEC      *alg;

    INT32      alg_num;      /* current algorithm */

```

```

        INT32      param_num;      /* current parameter number      */

615    BOOL        quit;            /* quit program              */
        BOOL        automation;    /* quit automated session?   */
        INT32      lsl_cmd;        /* Low Speed Link command    */

        FLOAT32     *ir;           /* for data preview          */
620    FLOAT32     *rd;
        INT32      preview_size = 650;

        INT32      ss_size;        /* snapshot size             */

625    INT32      i;
        INT32      j;
        BOOL      cal_ok;
        INT32      time;
        FLOAT32     f;

630    BUG1(num_algorithms);

        assert(num_algorithms <= MAX_ALGORITHMS);

635    mark_mem();
        alg      = get_mem(num_algorithms * sizeof(ALGORITHM_SPEC));

        /* now allocate structures for each algorithm */

640    for (i = 0; i < num_algorithms; i++) {
        alg[i].vp = get_mem(sizeof(SAT_PARAMS));
        alg[i].vr = get_mem(sizeof(SAT_RESULTS));
        alg[i].vhp = get_mem(sizeof(HR_PARAMS));
        alg[i].vhr = get_mem(sizeof(HR_RESULTS));
645    alg[i].cp = get_mem(sizeof(CALIB_PARAMS));
        alg[i].cr = get_mem(sizeof(CALIB_RESULTS));
    }

        /* init param structures */

650    for (i = 0; i < num_algorithms; i++) {
        for (j = 0; j < NUM_PARAMS; j++) {
            param_val[i][j] = params[j].def_val;
        }
655    }

        param_num = 0;
        alg_num = 0;
        quit = FALSE;

660    ir = get_mem(preview_size);
        rd = get_mem(preview_size);

        new_file(0);

665    set_alg(alg, num_algorithms);
        new_patient(alg, num_algorithms);

        store_parameters(num_algorithms); /* display initial parameters */
670    GDDCLI_FlushBuffer();

        CALIB_Init(alg[0].cp, alg[0].cr);
        do {
            GDDCLI_SendPacket(SPT_CalibrateMsg, 1.0);
            GDDCLI_FlushBuffer();
            cal_ok = CALIB_TwoPointCal(alg[0].cp, alg[0].cr);
            GDDCLI_Data(SPT_CalibResults, CALIB_RESULTS_SIZE,
675                                (FLOAT32 *) (alg[0].cr));
        } while (cal_ok == FALSE);
680    GDDCLI_SendPacket(SPT_CalibrateMsg, 0.0);

```

```

GDDCLI_FlushBuffer();

time = get_snapshot(ir, rd, preview_size);
store_raw(alg[0].vp, ir, rd, preview_size, time);
685 GDDCLI_FlushBuffer();

presentation_flag = FALSE;

while (quit == FALSE) {
690     GDDCLI_HighLight(SPT_SystemParam, param_num, alg_num,
                        param_val[alg_num][param_num]);
    GDDCLI_FlushBuffer();
    lsl_cmd = lsl_read();
695 GDDCLI_Normal(SPT_SystemParam, param_num, alg_num,
                  param_val[alg_num][param_num]);
    GDDCLI_FlushBuffer();

    switch(lsl_cmd) {
700     case NLT: /* left arrow */
        if (alg_num > 0) {
            alg_num -= 1;
        }
        break;
705     case NRT: /* right arrow */
        if (alg_num < num_algorithms-1) {
            alg_num += 1;
        }
        break;
710     case NUP: /* up arrow */
        if (param_num > 0) {
            param_num -= 1;
        }
        break;
715     case NDN: /* down arrow */
        if (param_num < NUM_PARAMS-1) {
            param_num += 1;
        }
        break;
720     case NINS :
        f = params[param_num].inc_val * 100.0;      /* HGK */
        goto kplus;

        case NPLUS:
        f = params[param_num].inc_val;      /* HGK */
730 #if (PLATFORM == 1)
        if(shift_key()) f*= 10.0;
    #endif

    kplus:
735     param_val[alg_num][param_num] += f;
        if(param_val[alg_num][param_num] > params[param_num].max_val)
            param_val[alg_num][param_num] = params[param_num].max_val;
        break;

740     case NDEL :
        f = params[param_num].inc_val * 100.0;      /* HGK */
        goto kminus;

        case NMINUS:
        f = params[param_num].inc_val;      /* HGK */
745 #if (PLATFORM == 1)
        if(shift_key()) f*= 10.0;
    #endif

```

```

750 kminus:
        param_val[alg_num][param_num] -= f;
        if(param_val[alg_num][param_num] < params[param_num].min_val)
            param_val[alg_num][param_num] = params[param_num].min_val;
        break;

755

        case NRTUP: /* page up */
            move_snapshot(-DATA_STEP);
            time = get_snapshot(ir, rd, preview_size);
760            store_raw(alg[0].vp, ir, rd, preview_size, time);
            GDDCLI_FlushBuffer();
            break;

        case NRTDN: /* page down */
765            move_snapshot( DATA_STEP);
            time = get_snapshot(ir, rd, preview_size);
            store_raw(alg[0].vp, ir, rd, preview_size, time);
            GDDCLI_FlushBuffer();
            break;

770

        case SCR: /* continuous operation */
            set_alg(alg, num_algorithms);
            ss_size = compute_snapshot_size(alg, num_algorithms);
/*
775            calibrate2(system_params.cal_target); */
            reset_file_num();
            new_file(0);

            automation = TRUE; /* 'continue automated session' */
            do
780            {
                time=0;
                new_patient(alg, num_algorithms);
                do
785                {
                    single_step(alg, num_algorithms, ss_size);
                    move_snapshot(USRI_AUTO_STEP);
                    /* mating get_snapshot() is in single_step.. */
                    if(lsl_status())
                        automation = FALSE; /* discontinue session */
790                }
                while (automation && !end_of_data());
                if(automation)
                    GDDCLI_NewFile();
            }
795            while (automation && !end_of_files());
            break;

        case SCS: /* single step */
            set_alg(alg, num_algorithms);
800            ss_size = compute_snapshot_size(alg, num_algorithms);
            single_step(alg, num_algorithms, ss_size);
            break;

        case SCC: /* "C" calibrate command */
805            break;

        case SCT: /* "T" test pattern command */
            test_pattern();
            break;

810

        case NESC: /* escape */
            quit = TRUE;
            break;

815

        case SCF: /* "F" new file command */
            new_file(1);

```

```

time = get_snapshot(ir, rd, preview_size);
store_raw(alg[0].vp, ir, rd, preview_size, time);
GDDCLI_FlushBuffer();;
820 break;

case SCB: /* "B" back file */
new_file(-1);
time = get_snapshot(ir, rd, preview_size);
825 store_raw(alg[0].vp, ir, rd, preview_size, time);
GDDCLI_FlushBuffer();
break;

case SCH :
830 param_val[alg_num][P_num_cells ]=2;
param_val[alg_num][P_search_size]=25;
break;

#if (PLATFORM == 1)
835 case SCW : presentation_flag = 1- presentation_flag;
break;
#endif

default: /* ignore unknown commands */
840 break;

} /* switch */

} /* while */

845 free_mem();
} /* user_interface */

```